

REST API Microversions

Versioning for Open Source services in Clouds

Sean Dague - Open Source Developer Advocate

@sdague

sean.dague@ibm.com / sean@dague.net

Nov 2nd 2017



Typical REST Versioning

Some time T1

GET /server/1

```
{
  "server": {
    "id": 1,
    "name": "myserver",
    "ips": ["10.42.100.1"]
  }
}
```

Some time T2

GET /server/1

```
{
  "server": {
    "id": 1,
    "name": "myserver",
    "description": "....",
    "ips": ["10.42.100.1"]
  }
}
```

Generally accepted rules:

- clients should not break on additional keys
- adding new attributes is backwards compatible
- there is only ever going to be one version publicly accessible*
- the API will never roll back once in production (i.e. description can't just disappear once it's out there)

Typical REST Versioning

Some time T1
GET /server/1

```
{
  "server": {
    "id": 1,
    "name": "myserver",
    "ips": ["10.42.100.1"]
  }
}
```

Some time T2
GET /server/1

```
{
  "server": {
    "id": 1,
    "name": "myserver",
    "description": "....",
    "ips": ["10.42.100.1"]
  }
}
```

Some time T3
GET /server/1

```
{
  "server": {
    "id": 1,
    "name": "myserver",
    "description": "....",
    "tags": ["db", "ha"],
    "ips": ["10.42.100.1"]
  }
}
```

All works very well for single vendor services

- Github
- AWS
- Meetup
- etc

Typical REST Versioning

Some time T1
GET /server/1

```
{  
  "server": {  
    "id": 1,  
    "name": "myserver",  
    "ips": ["10.42.100.1"]  
  }  
}
```

push to prod

Some time T2
GET /server/1

```
{  
  "server": {  
    "id": 1,  
    "name": "myserver",  
    "description": "....",  
    "ips": ["10.42.100.1"]  
  }  
}
```

push to prod

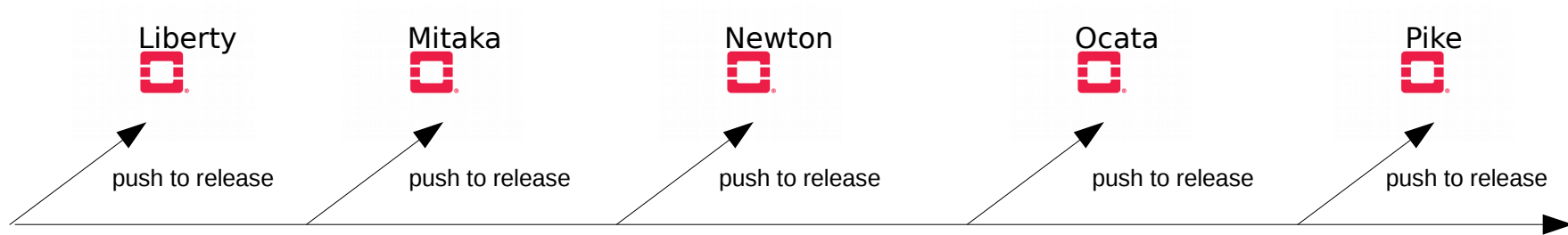
Some time T3
GET /server/1

```
{  
  "server": {  
    "id": 1,  
    "name": "myserver",  
    "description": "....",  
    "tags": ["db", "ha"],  
    "ips": ["10.42.100.1"]  
  }  
}
```

push to prod

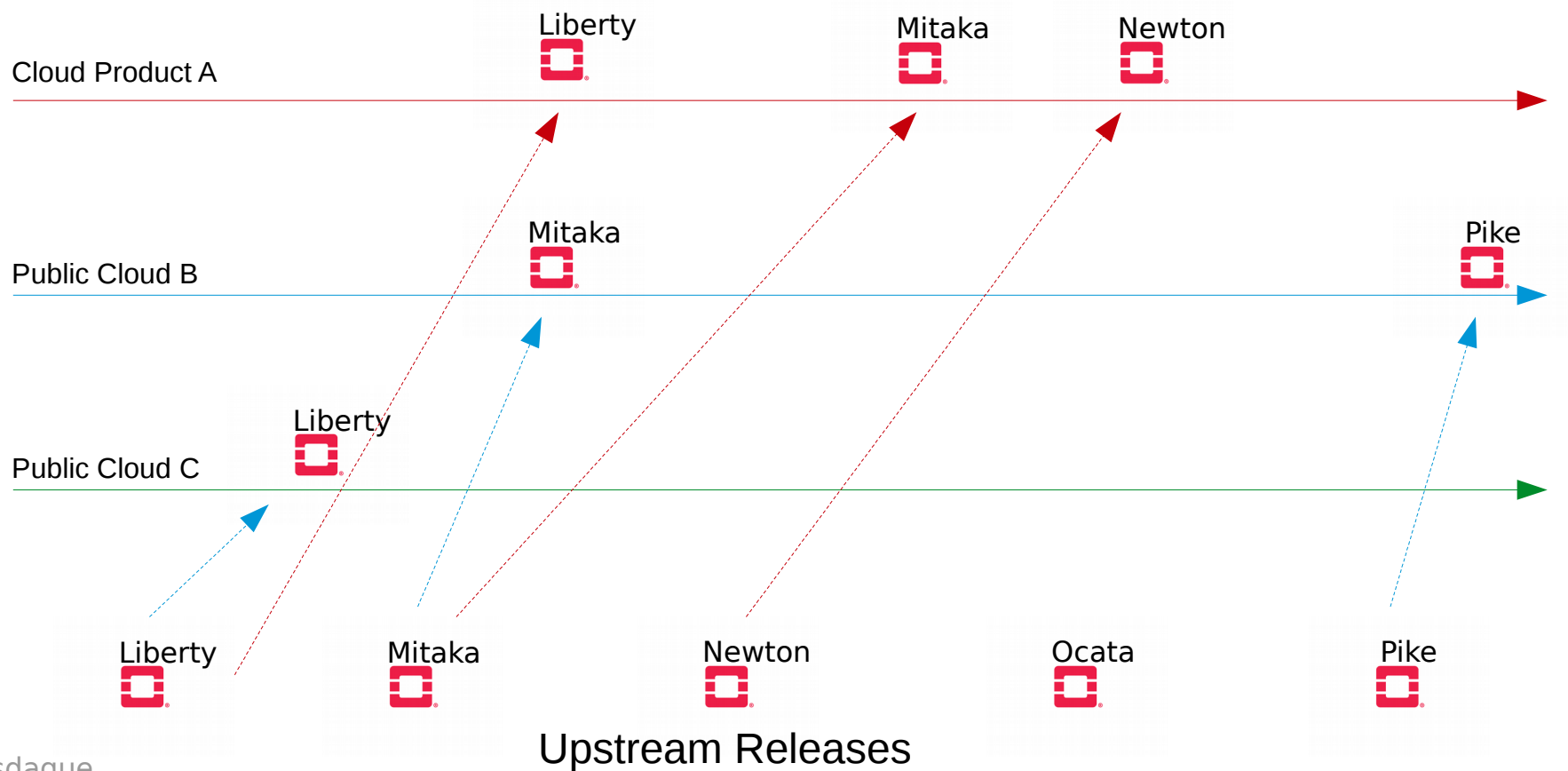
Development Workflow

Upstream Versioning

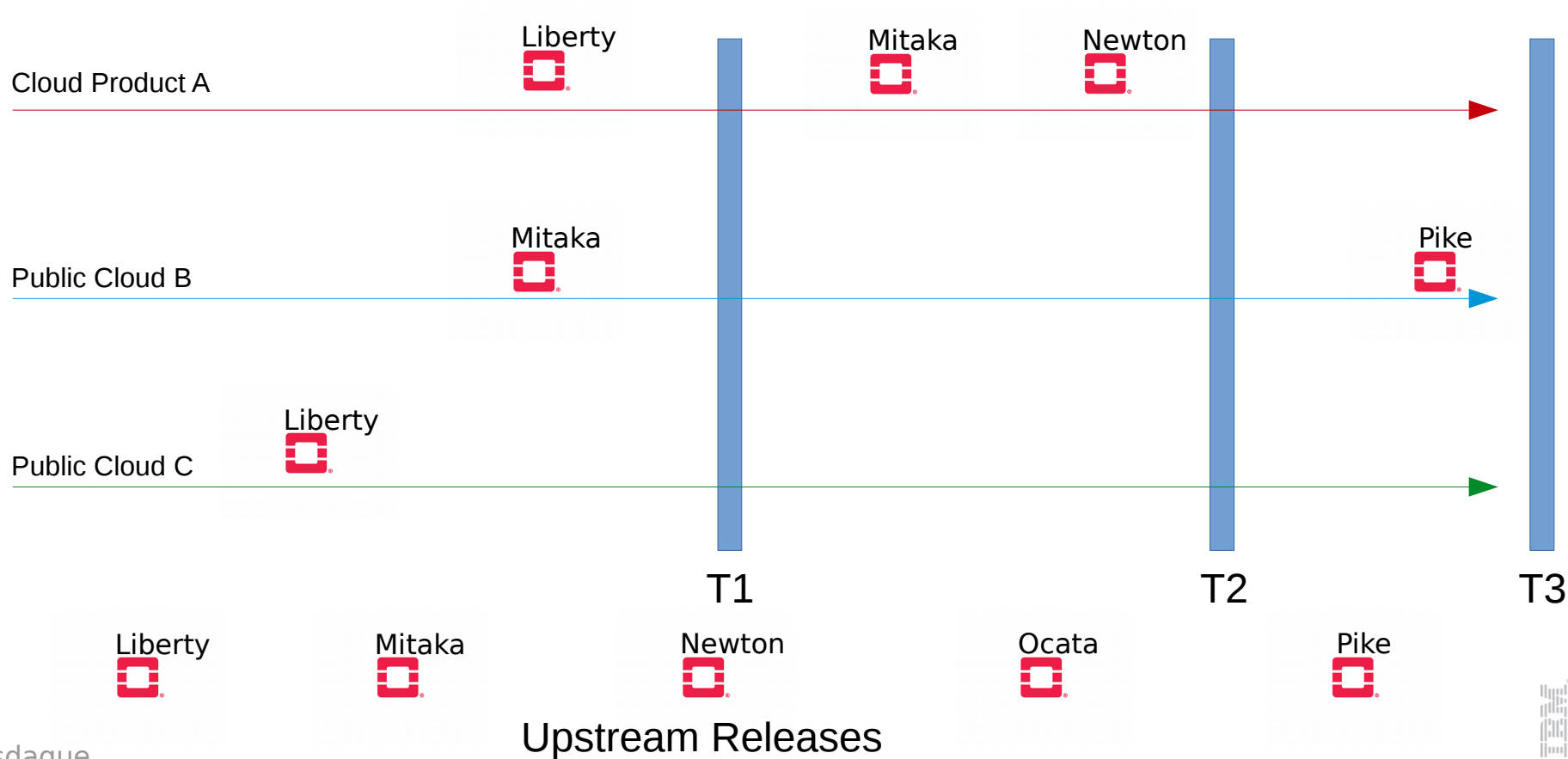


Upstream Development Workflow

Upstream Code going into Clouds and Products



Client Perspective



Client Experiencing $A \rightarrow B \rightarrow C$

Cloud A
GET /server/1

```
{
  "server": {
    "id": 1,
    "name": "myserver",
    "description": "....",
    "ips": ["10.42.100.1"]
  }
}
```

Cloud B
GET /server/1

```
{
  "server": {
    "id": 1,
    "name": "myserver",
    "description": "....",
    "tags": ["db", "ha"],
    "ips": ["10.42.100.1"]
  }
}
```

Cloud C
GET /server/1

```
{
  "server": {
    "id": 1,
    "name": "myserver",
    "ips": ["10.42.100.1"]
  }
}
```

Time's Arrow seems to go backwards.

Software is made to work on Cloud A, points at Cloud B still works, Cloud C is missing parameters leading to possibly confusing late failures.

**The Assumed Good Enough Rules
don't account for a
Different Team Developing the Software
than Deploying it**

REST API Microversions

Client Experiencing $A \rightarrow B \rightarrow C$

2.25

Cloud A
GET /server/1

```
{
  "server": {
    "id": 1,
    "name": "myserver",
    "description": "....",
    "ips": ["10.42.100.1"]
  }
}
```

2.40

Cloud B
GET /server/1

```
{
  "server": {
    "id": 1,
    "name": "myserver",
    "description": "....",
    "tags": ["db", "ha"],
    "ips": ["10.42.100.1"]
  }
}
```

Base

Cloud C
GET /server/1

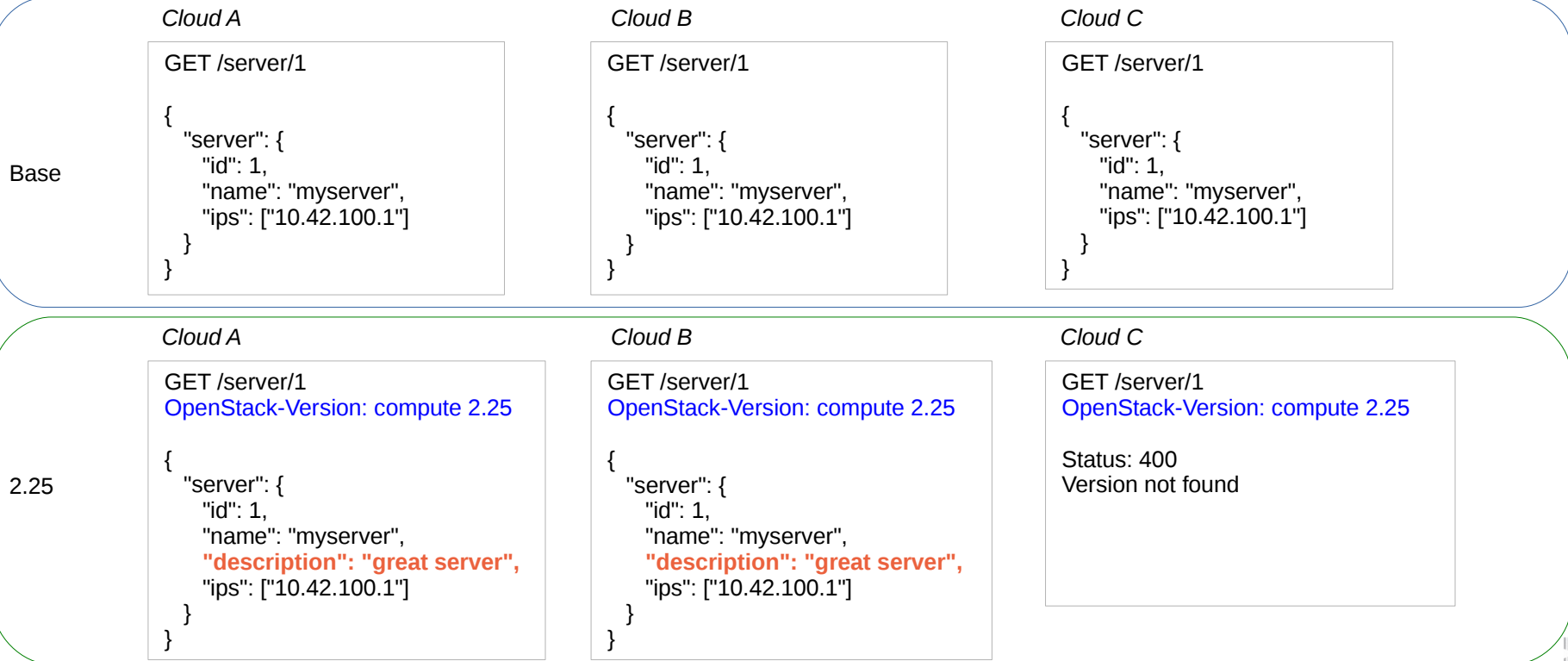
```
{
  "server": {
    "id": 1,
    "name": "myserver",
    "ips": ["10.42.100.1"]
  }
}
```

Time's Arrow seems to go backwards.

Software is made to work on Cloud A, points at Cloud B still works, Cloud C is missing parameters leading to possibly confusing late failures.

Client Experiencing A → B → C

At time T3 with Microversions



REST API Microversions

- Concept

- Inspired by HTTP Content negotiation
- Allow incremental feature roll out in API instead of waiting for major version bump
- Introduced in 2014 in OpenStack Nova / Ironic (expanded to 6 projects now)

- Technical Details

- Discoverable in root version document
- An explicit version header on each request
- Global incrementing version for entire service
- Defaults to minimum value
- Services continue to support old versions
- Hard 400 fail if version not found

- Blog writeup including personas - <https://dague.net/api-mv>



**Very long history of
getting here available
at bar later upon
request**



Questions for the Future

- Raising minimum versions?
- How much compat code do we need to keep?
- Machine specifications?

OpenStack Nova Compute:

GET /

```
{
  "versions": [
    {
      "id": "v2.0",
      "links": [
        {
          "href": "http://openstack.example.com/v2/",
          "rel": "self"
        }
      ],
      "status": "SUPPORTED",
      "version": "",
      "min_version": ""
    },
    {
      "id": "v2.1",
      "links": [
        {
          "href": "http://openstack.example.com/v2.1/",
          "rel": "self"
        }
      ],
      "status": "CURRENT",
      "version": "2.53",
      "min_version": "2.1"
    }
  ]
}
```

OpenStack and OpenAPI

OpenStack APIs vs OpenAPI

- Many OpenStack APIs evolved around WADL
- Attempt to retrofit OpenAPI definitions in 2015 to many core APIs, 2 key issues
 - Duplicate actions that don't map to Swagger/OpenAPI
 - `POST /servers/{id}/action`
`{"reboot": ""}`
 - `POST /servers/{id}/action`
`{"resize": {"flavor": "m1.large"}}`
 - Micro versions also don't really fit
 - Could they?



Parting Thoughts

THANKS!

- There are specific challenges to Open Source network consumable services being deployed in multiple organizationally controlled clouds
 - Tooling today largely doesn't consider that
- If we don't consider it, we make it harder to do non single vendor Open Source all the way to the edge
 - How do we prevent that?

<https://dague.net/api-mv>

Sean Dague

dague.net

sean.dague@ibm.com / sean@dague.net

@sdague

